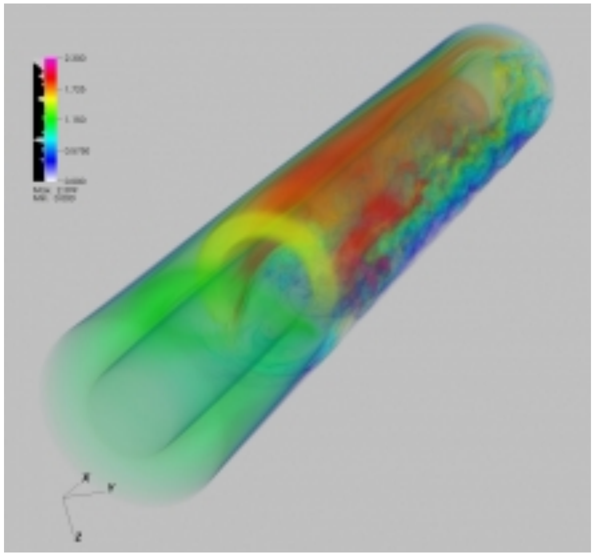
Nekbone

Scott Parker
Argonne
Leadership
Computing
Facility

What is Nek5000?

- **Spectral element CDF Solver for**
  - Unsteady incompressible Navier-Stokes
  - Low mach number flows
  - Heat transfer and species transport
  - Incompressible magnetohydrodynamics
- **Code:**
  - Open source
  - Written in Fortran 77 and C
  - MPI for parallelization
- **Features:**
  - Highly scalable, scales to over a million processes
  - High order spatial discretization using spectral elements
  - High order semi-implicit time stepping



What is Nekbone?

- **Nekbone is a benchmark derived from Nek5000**
  - Developed by Katherine Heisey, Paul Fischer, and Scott Parker
- **Solves 3D Poisson problem in rectangular geometry**
  - Spectral Element method with Conjugate Gradient linear solver
  - Large percentage of the work in Nek5000
  - Represents key kernels and operation mix from Nek5000:
    - matrix-matrix multiplication
    - inner products
    - nearest neighbor communication
    - MPI_Allreduce
- **Implemented using:**
  - Fortran 77, C, MPI, and OpenMP
- **Used for**
  - Exascale co-design activities: DOE FastForward, DesignForward
  - DOE machine acquisitions: CORAL systems
- **Available at**
  - https://cesar.mcs.anl.gov/content/software/thermal_hydraulics
  - https://asc.llnl.gov/CORAL-benchmarks/

Why Nekbone?

- **System designers need representative applications to study**
  - HPC has unique characteristics
- **In comparison to Nek5000 Nekbone is:**
  - More easily configurable
    - Number of spectral elements per rank
    - Polynomial order of element
  - Quicker to run
    - Run time is adjustable over a wide range
    - Typical run time is a few seconds
  - Allows multiple cases in one run
    - A range of elements can be specified
    - A range of polynomial orders can be specified
  - More easily instrumented
  - More easily modified
    - Has ~8K lines of code vs 60k lines of code for Nek5000
    - Re-implemented using other programming models: OpenMP, OpenACC, CUDA

Nekbone in a nutshell

cg() *[loop 1 -> numCGIterations]*

- **solveM()** *[z(i) = r(i)]*
- **glsc3()** *[inner product]*
  - **AllReduce()**
- **add2s1()** *[a(i) = c*a(i)+b(i)]*
- **ax()**
  - **ax_e()**
    - **local_grad3()** *[gradient]*
      - (3x) **mxm()**
    - **wr-ws-wt** *[wx(i) = f(g,ur,us,ut)]*
    - **local_grad3_t()** *[gradient]*
      - (3x) **mxm()**
      - (2x) **add2()**
  - **gs_op()** *[ptp communication]*
  - **add2s2()** *[a(i) = a(i) + c*b(i)]*
- **glsc3()** *[inner product]*
  - **AllReduce**
- **add2s2()** *[a(i) = a(i) + c*b(i)]*
- **add2s2()** *[a(i) = a(i) + c*b(i)]*
- **glsc3()** *[inner product]*
  - ***AllReduce***

- **Bandwidth Bound**
- **Compute Bound**
- **Network Bound**

gs_op()

- **gs_gather()** *[while, out[j] = out[j] + in[i]]*
- **pw_exec()**
  - **pw_exec_recvs()** *[MPI_Irecv]*
  - **gs_scatter()** *[while, out[j] = in[i]]*
  - **pw_exec_sends()** *[MPI_Isend]*
  - **comm_wait()** *[MPI_Waitall]*
  - **gs_gather()** *[while, out[j] = out[j] + in[i]]*
- **gs_scatter()** *[while, out[j] = in[i]]*

Nekbone Compute Performance Model

Routine	Routine	Routine	Routine	Data	Code	Loads	Stores	FPOps
solveM	copy			z,r	z(i)=r(i)	N	N	0
glsc3				r,c,z	t=t+r(i)*c(i)*z(i)	3N	0	3N
	gop	mpi_allreduce						
add2s1				p,z	p(i)=C*p(i)*z(i)	2N	N	2N
axi								
	ax_e							
		local_grad3						
			mxm	p,ur,dxm1		N	0	$2N_xN$
			mxm	p,us,dxtm1		0	0	$2N_xN$
			mxm	p,ut,dxtm1		0	0	$2N_xN$
		wrwswt		g,ur,us,ut	ur(i)=g(1,i)*ur(i)+g(2,i)*us(i)+g(3,i)*ut(i) us(i)=g(2,i)*ur(i)+g(4,i)*us(i)+g(5,i)*ut(i) ut(i)=g(3,i)*ur(i)+g(5,i)*us(i)+g(6,i)*ut(i)	6N	0	15N
		local_grad3_t						
			mxm	w,ur,dxtm1		0	0	$2N_x*N$
			mxm	t*,us,dxm1		0	0	$2N_x*N$
			add2	w,t*	w(i)=w(i)+t(i)	0	0	N
			mxm	t*,ut,dxm1		0	0	$2N_x*N$
			add2	w,t*	w(i)=w(i)+t(i)	0	N	N
	gs_op							
	add2s2			w,p	w(i)=w(i)+c*p(i)	2N	N	2N
	mask							
glsc3				w,c,p	t=t+w(i)*c(i)*p(i)	3N	0	3N
	gop	mpi_allreduce						
add2s2				x,p	x(i)=x(i)+C*p(i)	2N	N	2N
add2s2				r,w	r(i)=r(i)+C*w(i)	2N	N	2N
glsc3				r,c,r	t=t+r(i)*c(i)*r(i)	2N	0	3N

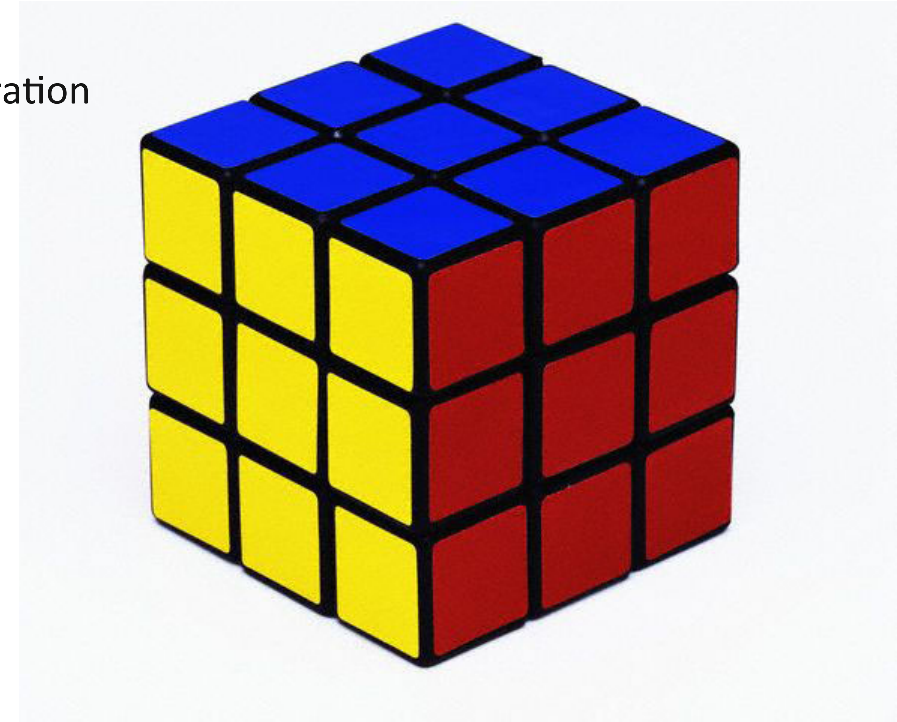
Nekbone Compute Performance Model

Routine	Av Time	% Time	Bytes Loaded/ it	Bytes Stored/it	FP Operations/ it	GB/s	Gflop/s	Est time	Err Ratio
Solver Time	1.38E+01								
rzero	2.16E-03	0.02%	0	67,108,864	0	31.04	0.00	0.0024	**1.11**
copy	5.67E-03	0.04%	67,108,864	67,108,864	0	23.65	0.00	0.0048	**0.84**
glsc3a	6.02E-03	0.04%	134,217,728	0	25,165,824	22.29	4.18	0.0048	**0.80**
gopa	2.81E-05	0.00%	0	0	0	0.00	0.00		
solveM	4.59E-01	3.33%	67,108,864	67,108,864	0	29.27	0.00	0.4793	**1.05**
glsc3b	8.80E-01	6.40%	201,326,592	0	25,165,824	22.87	2.86	0.7190	**0.82**
gopb	2.48E-03	0.02%	0	0	0	0.00	0.00		
add2s1	6.78E-01	4.93%	134,217,728	67,108,864	16,777,216	29.69	2.47	0.7190	**1.06**
localgrad3	2.89E+00	**20.98%**	67,108,864	0	805,306,368	2.32	27.89	0.3932	**0.14**
wrwswt	9.31E-01	6.77%	402,653,184	0	125,829,120	43.23	13.51	1.4380	**1.54**
localgradt	3.08E+00	**22.37%**	0	67,108,864	822,083,584	2.18	26.71	0.4014	**0.13**
gsop	1.21E+00	8.78%	0	0	0	0.00	0.00		
add2s2a	7.31E-01	5.31%	134,217,728	67,108,864	16,777,216	27.53	2.29	0.7190	**0.98**
glsc3c	8.77E-01	6.37%	201,326,592	0	25,165,824	22.95	2.87	0.7190	**0.82**
gopc	2.85E-03	0.02%	0	0	0	0.00	0.00		
add2s2b	6.86E-01	4.98%	134,217,728	67,108,864	16,777,216	29.35	2.45	0.7190	**1.05**
add2s2c	7.09E-01	5.15%	134,217,728	67,108,864	16,777,216	28.41	2.37	0.7190	**1.01**
glsc3d	5.98E-01	4.35%	134,217,728	0	25,165,824	22.44	4.21	0.4793	**0.80**
gopd	2.43E-03	0.02%	0	0	0	0.00	0.00		

Model Time	7.51
Actual Time	12.55
Error Ratio	**0.60**

Nekbone Communication

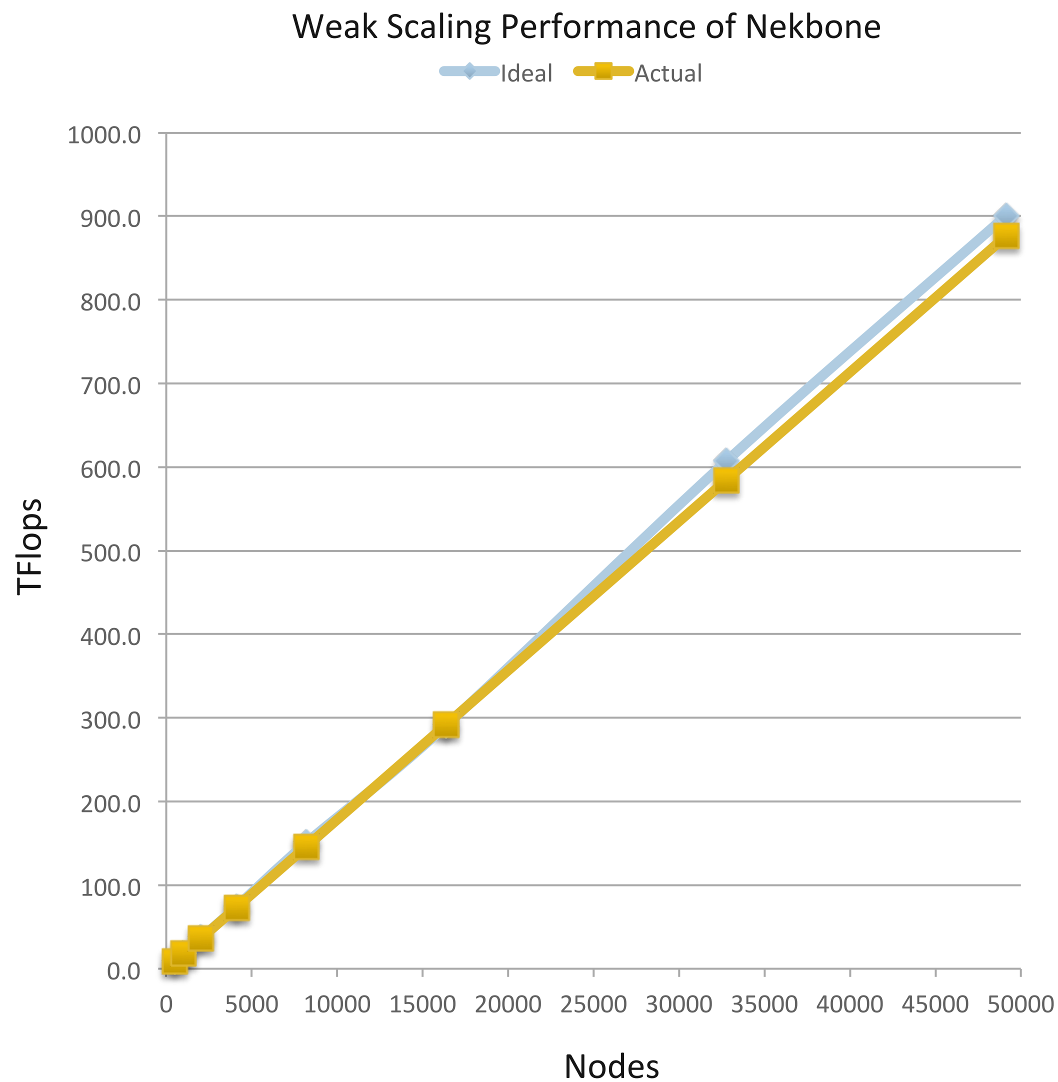
- **Point to Point Communication**
  - 26 send/receives per rank
    - 8 vertex values sent/received (8 Bytes per message, for 512x16 case)
    - 12 edges sent/received (128 Bytes per message, for 512x16 case)
    - 6 faces sent/received (16,384 Bytes per message, for 512x16 case)
- **Collective Communication**
  - Calls MPI_Allreduce 3 times per CG iteration
  - 8 Byte (1 double) reduction per call
  - 24 bytes per iteration



Nekbone Scaling on Mira

Grid Points per thread: ~10k
FLOP Rate: 9% of peak
Parallel Efficiency: 99%

Ranks	Threads	TFlops
512	64	9.5
1024	64	18.9
2048	64	36.9
4096	64	73.9
8192	64	150.5
16384	64	291.1
32768	64	606.9
49152	64	900.8



Typical Ratios on Representative BG/Q Runs

Ratio	Value
FLOPS/(Bytes Loaded & Stored)	0.94
Loaded Bytes/ Stored Bytes	4
FLOPS/AllReduce	158,000,000
FLOPS/Pt2Pt Byte	4,744
FLOPS/MPI-Message	9,111,545

Routine Type	Percentage
Memory Bound	45%
Compute Bound	35%
Point to Point Comm.	18%
Collective Comm.	2%

Adding OpenMP to Nekbone

- **Adding OpenMP:**
  - Relatively straightforward: 90% trivial, 10% required detailed understanding
  - Basic approach: partition element across threads
    - Easy:
      - Add a single OMP parallel region at top of cg() routine
      - Modify routines (add2s2, glsc3, axi, etc) to take a range of elements as an arg
      - Modify routines to use locally declared work arrays (ax_e)
    - A bit more complex:
      - Restructure gather/scatter maps for parallel execution
      - Add synchronization and barriers around communication operations (gs_aux, pw_exec)
- **Impact:**
  - Little impact on compute performance
  - Little impact on memory usage
  - Some impact on communication performance, most noticeable at large scale
    - Eliminates some data copies to/from MPI buffers
    - Fewer messages sent
  - Provides opportunity to overlap communication and computation

Nekbone on KNL

- **Nekbone is up and running on KNL**
  - Simulations and estimates of performance based on KNL specs
  - Run on pre-release KNL hardware
  - Performance as expected based on compute performance model
  - Tuning use of AVX-512 instructions
    - Utilizing LIBXSMM for matrix multiplication

Next Steps

- **KNL Optimization**
- **Programming Models**
  - CUDA
  - OpenMP 4
  - OpenACC
  - RAJA, Kokkos
- **Overlap computation and communication**
  - Communication kernel can be rewritten to send messages as soon as they are ready
  - Element updates can be re-ordered to update process boundary elements first
  - Process interior elements can updated simultaneous with communication operations